```
with BASE,
     EXCEPTIONS,
     TYP,
     OPS,
     CHECK,
     FAULT;

separate (EXECUTION_UNIT.OBJECT_OP)
package body INT_OP is

    use BASE;

    function GET_ONE_ARG return STACKS.CONTROL_QUAD
                                        (OF_KIND => STACKS.INTEGER_VAR) is
        OPERAND : STACKS.CONTROL_QUAD
                                  (OF_KIND => STACKS.INTEGER_VAR);
    begin
        OPERAND := CONTROL.POP ();
        CHECK.CLASS   (OPERAND, STACKS.INTEGER_VAR);
        CHECK.PRIVACY (OPERAND);
        return OPERAND;
    end GET_ONE_ARG;


    function GET_ONE_VALUE return BASE.INT is
    begin
        return GET_ONE_ARG().DATA_ITEM.VAL.INT_VAL;
    end GET_ONE_VALUE;


    procedure GET_TWO_ARGS (LEFT_ARG   : out STACKS.CONTROL_QUAD;
                            RIGHT_ARG  : out STACKS.CONTROL_QUAD) is
    begin
        RIGHT_ARG := CONTROL.POP ();
        LEFT_ARG  := CONTROL.POP ();

        CHECK.CLASS (RIGHT_ARG, LEFT_ARG, STACKS.INTEGER_VAR);
        CHECK.PRIVACY (RIGHT_ARG);
        CHECK.PRIVACY (LEFT_ARG);
    end GET_TWO_ARGS;


    procedure GET_TWO_VALUES (LEFT_VALUE  : out BASE.INT;
                              RIGHT_VALUE : out BASE.INT) is
        LEFT_ARG  : STACKS.CONTROL_QUAD
                          (OF_KIND => STACKS.INTEGER_VAR);
        RIGHT_ARG : STACKS.CONTROL_QUAD
                          (OF_KIND => STACKS.INTEGER_VAR);
    begin
        GET_TWO_ARGS (LEFT_ARG, RIGHT_ARG);
        LEFT_VALUE  := LEFT_ARG.DATA_ITEM.VAL.INT_VAL;
        RIGHT_VALUE := RIGHT_ARG.DATA_ITEM.VAL.INT_VAL;
    end GET_TWO_VALUES;
```

```
PROCEDURE DO_EQUAL_OP (OPERATOR : CODE.OPERATOR) is
    RIGHT_ARG : STACKS.CONTROL_QUAD
                              (OF_KIND => STACKS.INTEGER_VAR);
    LEFT_ARG  : STACKS.CONTROL_QUAD
                              (OF_KIND => STACKS.INTEGER_VAR);
    TEST      : BOOLEAN;
begin
    RIGHT_ARG := CONTROL.POP ();
    LEFT_ARG  := CONTROL.POP ();

    CHECK.CLASS (RIGHT_ARG, LEFT_ARG, STACKS.INTEGER_VAR);
    CHECK.LIMITATION (RIGHT_ARG);
    CHECK.LIMITATION (LEFT_ARG);

    case OPERATOR is
        when CODE.EQUAL_OP     =>
            TEST := (LEFT_ARG.DATA_ITEM.VAL.INT_VAL
                              = RIGHT_ARG.DATA_ITEM.VAL.INT_VAL);

        when CODE.NOT_EQUAL_OP =>
            TEST := (LEFT_ARG.DATA_ITEM.VAL.INT_VAL
                              /= RIGHT_ARG.DATA_ITEM.VAL.INT_VAL);

        when others =>
             FAULT.HANDLE (EXCEPTIONS.INSTRUCTION_ERROR);
    end case;

    CONTROL.PUSH (OPS.BOOL_QUAD_WORD (TEST));
end DO_EQUAL_OP;


PROCEDURE DO_REL_OP (OPERATOR : CODE.OPERATOR) is
    LEFT_VALUE  : BASE.INT;
    RIGHT_VALUE : BASE.INT;
    TEST        : BOOLEAN;
begin
    GET_TWO_VALUES (LEFT_VALUE, RIGHT_VALUE);
    case OPERATOR is
        when CODE.LESS_OP          => TEST := (LEFT_VALUE <  RIGHT_VALUE);
        when CODE.LESS_EQUAL_OP    => TEST := (LEFT_VALUE <= RIGHT_VALUE);
        when CODE.GREATER_OP       => TEST := (LEFT_VALUE >  RIGHT_VALUE);
        when CODE.GREATER_EQUAL_OP => TEST := (LEFT_VALUE >= RIGHT_VALUE);
        when others                => ERROR.ILLEGAL_INSTRUCTION;
    end case;

    CONTROL.PUSH (OPS.BOOL_QUAD_WORD (TEST));
end DO_REL_OP;
```

```
procedure DO_FIRST_OP is
    OPERAND : STACKS.CONTROL_QUAD (OF_KIND => STACKS.INTEGER_VAR);
    BOUNDS  : STACKS.TYPE_QUAD    (OF_KIND => STACKS.SCALAR_BOUNDS);
begin
    OPERAND := GET_ONE_ARG();
    BOUNDS  := TYP.READ (OPERAND.DATA_TYPE.PATH);
    OPERAND.DATA_ITEM.VAL.INT_VAL := BOUNDS.MINIMUM.INT_VAL;
    CONTROL.PUSH (OPERAND);
end DO_FIRST_OP;


procedure DO_LAST_OP is
    OPERAND : STACKS.CONTROL_QUAD (OF_KIND => STACKS.INTEGER_VAR);
    BOUNDS  : STACKS.TYPE_QUAD    (OF_KIND => STACKS.SCALAR_BOUNDS);
begin
    OPERAND := GET_ONE_ARG();
    BOUNDS  := TYP.READ (OPERAND.DATA_TYPE.PATH);
    OPERAND.DATA_ITEM.VAL.INT_VAL := BOUNDS.MAXIMUM.INT_VAL;
    CONTROL.PUSH (OPERAND);
end DO_LAST_OP;
```

```
procedure DO_INTERVAL_OP (OPERATOR : CODE.OPERATOR) is
    CONTROL_BOUNDS : STACKS.CONTROL_QUAD (OF_KIND => STACKS.INTEGER_VAR);
    TYPE_BOUNDS    : STACKS.TYPE_QUAD    (OF_KIND => STACKS.SCALAR_BOUNDS);
    MINIMUM        : BASE.INT;
    MAXIMUM        : BASE.INT;
    TEST_VALUE     : BASE.INT;
    TEST           : BOOLEAN;
begin
    case OPERATOR is
        when CODE.IN_TYPE_OP  | CODE.NOT_IN_TYPE_OP  =>
            CONTROL_BOUNDS := GET_ONE_ARG ();
            TYPE_BOUNDS    := TYP.READ (CONTROL_BOUNDS.DATA_TYPE.PATH);
            MAXIMUM        := TYPE_BOUNDS.MAXIMUM.INT_VAL;
            MINIMUM        := TYPE_BOUNDS.MINIMUM.INT_VAL;

        when CODE.IN_RANGE_OP | CODE.NOT_IN_RANGE_OP =>
            CONTROL_BOUNDS := GET_ONE_ARG ();
            CHECK.CLASS (CONTROL_BOUNDS, STACKS.INTEGER_VAR);
            MAXIMUM        := CONTROL_BOUNDS.DATA_ITEM.VAL.INT_VAL;
            CONTROL_BOUNDS := GET_ONE_ARG ();
            CHECK.CLASS (CONTROL_BOUNDS, STACKS.INTEGER_VAR);
            MINIMUM        := CONTROL_BOUNDS.DATA_ITEM.VAL.INT_VAL;

        when others =>
            ERROR.ILLEGAL_INSTRUCTION;
    end case;

    TEST_VALUE := GET_ONE_VALUE ();

    case OPERATOR is
        when CODE.IN_TYPE_OP | CODE.IN_RANGE_OP =>
            TEST := (MINIMUM <= TEST_VALUE) and (TEST_VALUE <= MAXIMUM);

        when CODE.NOT_IN_TYPE_OP | CODE.NOT_IN_RANGE_OP =>
            TEST := (TEST_VALUE < MINIMUM) or (MAXIMUM < TEST_VALUE);

        when others =>
            null;
    end case;

    CONTROL.PUSH (OPS.BOOL_QUAD_WORD (TEST));
end DO_INTERVAL_OP;
```

```
procedure DO_RANGE_OP (OPERATOR : CODE.OPERATOR) is
    OPERAND      : STACKS.CONTROL_QUAD (OF_KIND => STACKS.INTEGER_VAR);
    BOUNDS       : STACKS.TYPE_QUAD    (OF_KIND => STACKS.SCALAR_BOUNDS);
    BOUND_VALUE  : BASE.INT;
    TEST         : BOOLEAN;
begin
    OPERAND    := GET_ONE_ARG ();
    BOUNDS     := TYP.READ (OPERAND.DATA_TYPE.PATH);
    case OPERATOR is
        when CODE.START_RANGE_OP => BOUND_VALUE := BOUNDS.MINIMUM.INT_VAL;
        when CODE.END_RANGE_OP   => BOUND_VALUE := BOUNDS.MAXIMUM.INT_VAL;
        when others              => ERROR.ILLEGAL_INSTRUCTION;
    end case;

    if OPERAND.DATA_ITEM.VAL.INT_VAL = BOUND_VALUE then
        TEST := TRUE;
    else
        CONTROL.PUSH (OPERAND);
        TEST := FALSE;
    end if;

    CONTROL.PUSH (OPS.BOOL_QUAD_WORD (TEST));
end DO_RANGE_OP;
```

```
procedure DO_SUCC_OP is
    OPERAND : STACKS.CONTROL_QUAD (OF_KIND => STACKS.INTEGER_VAR);
    BOUNDS  : STACKS.TYPE_QUAD    (OF_KIND => STACKS.SCALAR_BOUNDS);
    VALUE   : BASE.INT;
begin
    OPERAND := GET_ONE_ARG ();
    BOUNDS  := TYP.READ (OPERAND.DATA_TYPE.PATH);
    VALUE   := OPERAND.DATA_ITEM.VAL.INT_VAL + 1;

    if  VALUE > BOUNDS.MAXIMUM.INT_VAL then
        FAULT.HANDLE (EXCEPTIONS.CONSTRAINT_ERROR);
    end if;

    OPERAND.DATA_ITEM.VAL.INT_VAL := VALUE;
    CONTROL.PUSH (OPERAND);
end DO_SUCC_OP;


procedure DO_PRED_OP is
    OPERAND : STACKS.CONTROL_QUAD (OF_KIND => STACKS.INTEGER_VAR);
    BOUNDS  : STACKS.TYPE_QUAD    (OF_KIND => STACKS.SCALAR_BOUNDS);
    VALUE   : BASE.INT;
begin
    OPERAND := GET_ONE_ARG ();
    BOUNDS  := TYP.READ (OPERAND.DATA_TYPE.PATH);
    VALUE   := OPERAND.DATA_ITEM.VAL.INT_VAL - 1;

    if  VALUE > BOUNDS.MINIMUM.INT_VAL then
        FAULT.HANDLE (EXCEPTIONS.CONSTRAINT_ERROR);
    end if;

    OPERAND.DATA_ITEM.VAL.INT_VAL := VALUE;
    CONTROL.PUSH (OPERAND);
end DO_PRED_OP;


procedure DO_INCREMENT_OP is
    OPERAND : STACKS.CONTROL_QUAD (OF_KIND => STACKS.INTEGER_VAR);
begin
    OPERAND := GET_ONE_ARG ();
    OPERAND.DATA_ITEM.VAL.INT_VAL := OPERAND.DATA_ITEM.VAL.INT_VAL + 1;
    CONTROL.PUSH (OPERAND);
end DO_INCREMENT_OP;


procedure DO_DECREMENT_OP is
    OPERAND : STACKS.CONTROL_QUAD (OF_KIND => STACKS.INTEGER_VAR);
begin
    OPERAND := GET_ONE_ARG ();
    OPERAND.DATA_ITEM.VAL.INT_VAL := OPERAND.DATA_ITEM.VAL.INT_VAL - 1;
    CONTROL.PUSH (OPERAND);
end DO_DECREMENT_OP;
```

```
procedure DO_UNARY_MINUS_OP is
    OPERAND : STACKS.CONTROL_QUAD (OF_KIND => STACKS.INTEGER_VAR);
begin
    OPERAND := GET_ONE_ARG ();
    OPERAND.DATA_ITEM.VAL.INT_VAL := - OPERAND.DATA_ITEM.VAL.INT_VAL;
    CONTROL.PUSH (OPERAND);
end DO_UNARY_MINUS_OP;



procedure DO_ABS_OP  is
    OPERAND : STACKS.CONTROL_QUAD (OF_KIND => STACKS.INTEGER_VAR);
begin
    OPERAND := GET_ONE_ARG ();
    if  OPERAND.DATA_ITEM.VAL.INT_VAL < 0  then
        OPERAND.DATA_ITEM.VAL.INT_VAL := - OPERAND.DATA_ITEM.VAL.INT_VAL;
    end if;
    CONTROL.PUSH (OPERAND);
end DO_ABS_OP;


procedure ARITHMETIC_OP (OPERATOR : CODE.OPERATOR)  is    -- binary op's
    LEFT_OPERAND  : STACKS.CONTROL_QUAD (OF_KIND => STACKS.INTEGER_VAR);
    RIGHT_OPERAND : STACKS.CONTROL_QUAD (OF_KIND => STACKS.INTEGER_VAR);
    LEFT_VALUE    : BASE.INT;
    RIGHT_VALUE   : BASE.INT;
    RESULT_VAL    : BASE.INT;
begin
    GET_TWO_ARGS (LEFT_OPERAND, RIGHT_OPERAND);
    LEFT_VALUE  := LEFT_OPERAND.DATA_ITEM.VAL.INT_VAL;
    RIGHT_VALUE := RIGHT_OPERAND.DATA_ITEM.VAL.INT_VAL;

    case OPERATOR is
        when CODE.PLUS_OP   =>
            RESULT_VAL := LEFT_VALUE + RIGHT_VALUE;
        when CODE.MINUS_OP  =>
            RESULT_VAL := LEFT_VALUE - RIGHT_VALUE;
        when CODE.TIMES_OP  =>
            RESULT_VAL := LEFT_VALUE * RIGHT_VALUE;
        when CODE.DIVIDE_OP =>
            RESULT_VAL := LEFT_VALUE / RIGHT_VALUE;
        when CODE.MOD_OP    =>
            RESULT_VAL := LEFT_VALUE mod RIGHT_VALUE;
        when CODE.REM_OP    =>
            RESULT_VAL :=
                LEFT_VALUE - ((LEFT_VALUE /RIGHT_VALUE) * RIGHT_VALUE);
        when others         =>
            ERROR.ILLEGAL_INSTRUCTION;
    end case;

    LEFT_OPERAND.DATA_ITEM.VAL.INT_VAL := RESULT_VAL;
    CONTROL.PUSH (LEFT_OPERAND);
end ARITHMETIC_OP;
```

```
procedure DO_DUPLICATE_OP is
    OPERAND : STACKS.CONTROL_QUAD (OF_KIND => STACKS.INTEGER_VAR);
begin
    OPERAND := GET_ONE_ARG();
    CONTROL.PUSH (OPERAND);
    CONTROL.PUSH (OPERAND);
end DO_DUPLICATE_OP;


function COMPLETE_DECLARATION (AT_SITE : STACKS.TYPE_REFERENCE)
                                                    return STACKS.VALUE_SIZE is
    PARENT         : STACKS.CONTROL_QUAD (OF_KIND => STACKS.INTEGER_VAR);
    PARENT_BOUNDS  : STACKS.TYPE_QUAD    (OF_KIND => STACKS.SCALAR_BOUNDS);
    MINIMUM_QUAD   : STACKS.CONTROL_QUAD (OF_KIND => STACKS.INTEGER_VAR);
    MAXIMUM_QUAD   : STACKS.CONTROL_QUAD (OF_KIND => STACKS.INTEGER_VAR);
    MINIMUM        : STACKS.VALUE;
    MAXIMUM        : STACKS.VALUE;
begin
    PARENT  := GET_ONE_ARG ();
    PARENT_BOUNDS := TYP.READ (PARENT.DATA_TYPE.PATH);

    GET_TWO_ARGS (LEFT_ARG => MINIMUM_QUAD, RIGHT_ARG => MAXIMUM_QUAD);
    MINIMUM := MINIMUM_QUAD.DATA_ITEM.VAL;
    MAXIMUM := MAXIMUM_QUAD.DATA_ITEM.VAL;

    if MAXIMUM.INT_VAL < MINIMUM.INT_VAL - 1
            or  MINIMUM.INT_VAL < PARENT_BOUNDS.MINIMUM.INT_VAL
            or  PARENT_BOUNDS.MAXIMUM.INT_VAL < MAXIMUM.INT_VAL  then
        FAULT.HANDLE (EXCEPTIONS.CONSTRAINT_ERROR);
    else
        TYP.WRITE (TARGET => AT_SITE,
                   VALUE  => STACKS.TYPE_QUAD'
                                    (OF_KIND => STACKS.SCALAR_BOUNDS,
                                     MINIMUM => MINIMUM,
                                     MAXIMUM => MAXIMUM));
        return OPS.COMPUTE_SCALAR_SIZE (MINIMUM, MAXIMUM);
    end if;
end COMPLETE_DECLARATION;
```

```
procedure DO_COMPLETE_OP is
    REF_VAR    : STACKS.CONTROL_QUAD (OF_KIND => STACKS.REFERENCE_VAR);
    VAR_SITE   : STACKS.CONTROL_REFERENCE;
    TYPE_VAR   : STACKS.CONTROL_QUAD (OF_KIND => STACKS.INTEGER_VAR);
begin
    REF_VAR    := CONTROL.POP ();
    CHECK.CLASS (REF_VAR, STACKS.REFERENCE_VAR);
    VAR_SITE   := REF_VAR.VARIABLE_REF;

    TYPE_VAR   := CONTROL.READ (TARGET => VAR_SITE);
    TYPE_VAR.DATA_TYPE.ITEM_SIZE :=
                      COMPLETE_DECLARATION (TYPE_VAR.DATA_TYPE.PATH);

    CONTROL.WRITE (TARGET => VAR_SITE,
                   VALUE  => TYPE_VAR);
end DO_COMPLETE_OP;


procedure TYPE_DECL (VISIBILITY : CODE.VISIBILITY;
                     LIMITATION : CODE.LIMITATION;
                     COMPLETE   : CODE.DECL_STATE)  is
    TYPE_SITE : STACKS.TYPE_REFERENCE;
    ITEM_SIZE : STACKS.VALUE_SIZE;
    NEW_TYPE  : STACKS.CONTROL_QUAD (OF_KIND => STACKS.INTEGER_VAR);
begin
    TYPE_SITE := TYP.ALLOCATE (SIZE => 1);
    if COMPLETE = CODE.COMPLETE  then
        ITEM_SIZE := COMPLETE_DECLARATION (TYPE_SITE);
    else
        ITEM_SIZE := STACKS.VALUE_SIZE'LAST;
    end if;

    NEW_TYPE := STACKS.CONTROL_QUAD'
                              (OF_KIND   => STACKS.INTEGER_VAR,
                               DATA_ITEM => STACKS.DATUM'
                                   (SORT => STACKS.IS_VAL,
                                    VAL  => STACKS.VALUE'
                                        (OF_KIND  => STACKS.INTEGER_VAR,
                                         INT_VAL  => BASE.INT'FIRST)),
                               DATA_TYPE => STACKS.TYPE_LINK'
                                        (OF_KIND    => STACKS.INTEGER_VAR,
                                         PATH       => TYPE_SITE,
                                         VISIBILITY => VISIBILITY,
                                         LIMITATION => LIMITATION,
                                         ITEM_SIZE  => ITEM_SIZE));
    CONTROL.PUSH (NEW_TYPE);
end TYPE_DECL;


procedure VAR_DECL (VISIBILITY : CODE.VISIBILITY)  is
    NEW_VAR : STACKS.CONTROL_QUAD;
begin
    NEW_VAR := CONTROL.POP ();
    NEW_VAR.DATA_ITEM.VAL.INT_VAL := BASE.INT'FIRST;
    NEW_VAR.DATA_TYPE.VISIBILITY  := VISIBILITY;

    CONTROL.PUSH (NEW_VAR);
end VAR_DECL;
```

```
    procedure DISPATCH (OPERATOR : CODE.OPERATOR) is
    begin
        case OPERATOR is

            when CODE.EQUALITY_OP     => DO_EQUAL_OP (OPERATOR);

            when CODE.RELATIONAL_OP   => DO_REL_OP (OPERATOR);

            when CODE.FIRST_OP        => DO_FIRST_OP;

            when CODE.LAST_OP         => DO_LAST_OP;

            when CODE.MEMBER_OP       => DO_INTERVAL_OP (OPERATOR);

            when CODE.RANGE_OP        => DO_RANGE_OP (OPERATOR);

            when CODE.SUCC_OP         => DO_SUCC_OP;

            when CODE.PRED_OP         => DO_PRED_OP;

            when CODE.INCREMENT_OP    => DO_INCREMENT_OP;

            when CODE.DECREMENT_OP    => DO_DECREMENT_OP;

            when CODE.ABS_OP          => DO_ABS_OP;

            when CODE.UNARY_MINUS_OP  => DO_UNARY_MINUS_OP;

            when CODE.ARITHMETIC_OP   => ARITHMETIC_OP (OPERATOR);

            when CODE.COMPLETE_OP     => DO_COMPLETE_OP;

            when CODE.DUPLICATE_OP    =>  DO_DUPLICATE_OP;

            when others               => ERROR.ILLEGAL_INSTRUCTION;
        end case;
    end DISPATCH;


end INT_OP;
```